Ps7

Video link is:
https://www.youtube.com/watch?v=AF3VA74lRTM&t=0s&index=6&list=PL1WxStBxgFL-uERja547HstCQPLWaUIl2
Code link is :
https://github.com/chenhuiyang1994/EECS376_ps7

Part 1: edit traj_builder

To complete the *braking_traj function*, refer to the ramp down of other traj_builder function :

```
//compute trajectory corresponding to applying max prudent decel to halt
void TrajBuilder::build_braking_traj(geometry_msgs::PoseStamped start_pose,
        std::vector<nav_msgs::Odometry> &vec_of_states) {
//refer to TrajBuilder::build_triangular_travel_traj
        vec_of_states.clear();
        nav_msgs::Odometry des_state;
        double omega_des;
        double psi_start = convertPlanarQuat2Psi(start_pose.pose.orientation);
        double psi_des = psi_start;
        double x_start = start_pose.pose.position.x;
        double y_start = start_pose.pose.position.y;
        double x_des = x_start; //start from here
        double y_des = y_start;
        double speed_des = 0.0;
        des_state.twist.twist.angular.z = 0.0; //omega_des; will not change
        des_state.pose.pose.orientation = convertPlanarPsi2Quaternion(psi_des); //constant
// orientation of des_state will not change; only position and twist
        int npts_ramp=1/dt_;
        for (int i = 0; i < npts_ramp; i++) {
        speed_des = accel_max_*dt_;
        des_state.twist.twist.linear.x = speed_des;//update speed
                //update positions
        x_des += -accel_max_ * dt_ * dt_ * cos(psi_des); //Euler one-step integration
        y_des += -accel_max_ * dt_ * dt_ * sin(psi_des); //Euler one-step integration
        des_state.pose.pose.position.x = x_des;
        des_state.pose.pose.position.y = y_des;
        omega_des += -alpha_max_*dt_; //Euler one-step integration
        des_state.twist.twist.angular.z = omega_des;
        psi_des += -alpha_max_*dt_ *dt_; //Euler one-step integration
        des_state.pose.pose.orientation = convertPlanarPsi2Quaternion(psi_des);
        vec_of_states.push_back(des_state);
        }
        des_state.twist.twist = halt_twist_;
        vec_of_states.push_back(des_state);
        ROS_INFO("%f",speed_des);
        ROS_INFO("Braking traj complete");
}
```

This shall bring down the velocity to 0 once the function is called.

Part 2: edit pub_des_state.cpp

The e-stop braking mechanism shall be very similar to the lidar alarm braking one.
In the code provided, there are two callBack function for e-stop to change the state of the machine, one is estopServiceCallback which will set the e-stop trigger to be true, this will lead the machine to the state of Halting which will call the braking function I just added to traj_builder (pub_des_state is build depend on traj_builder). The other is clearEstopServiceCallback which will cleans the e-stop trigger and put the machine back to the state DONE_W_SUBGOAL which will let the machine to keep

moving if there are still any unfinished jobs. So a simple way to mimic the estop halting mechanism is to these two callBacks for lidar alarm.

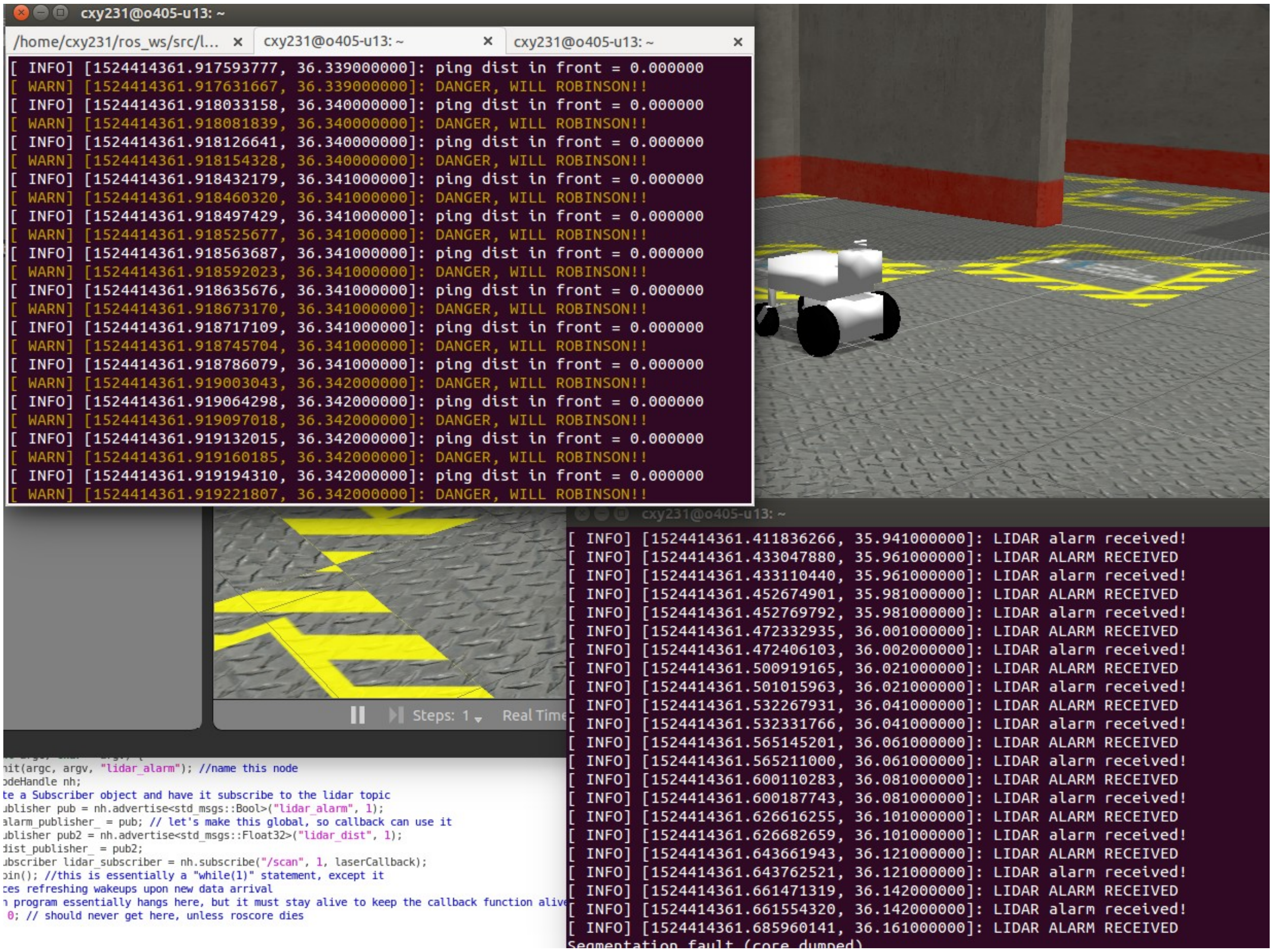
For the simulation to work and to keep using my lidar alarm code from assignment 2, I instead write a alarmCallback function which simply hear from lidar topic. Once the lidar alarm is heard, it will change the robot state to Halting.

```
//===============add lidar Cb function=================
void alarmCallback(const std_msgs::Bool& alarm_msg)
{
  g_lidar_alarm = alarm_msg.data; //make the alarm status global, so main() can use it
  if (g_lidar_alarm) {
     ROS_INFO("LIDAR alarm received!");
  }
}

void DesStatePublisher::pub_next_state() {
    // first test if an e-stop has been triggered
    if (e_stop_trigger_) {
        e_stop_trigger_ = false; //reset trigger
        //compute a halt trajectory
        trajBuilder_.build_braking_traj(current_pose_, des_state_vec_);
        motion_mode_ = HALTING;
        traj_pt_i_ = 0;
        npts_traj_ = des_state_vec_.size();
    }
    //or if an e-stop has been cleared
    if (e_stop_reset_) {
        e_stop_reset_ = false; //reset trigger
        if (motion_mode_ != E_STOPPED) {
            ROS_WARN("e-stop reset while not in e-stop mode");
        }
        else {
            motion_mode_ = DONE_W_SUBGOAL; //this will pick up where left off
            }
        }
 // lidar alarm
   if(g_lidar_alarm){
        motion_mode_ = HALTING;//a state machine
        ROS_INFO("LIDAR ALARM RECEIVED");
    }

    //state machine; results in publishing a new desired state
    switch (motion_mode_) {
        case E_STOPPED: //this state must be reset by a service
            desired_state_publisher_.publish(halt_state_);
            break;
```

But I run into a problem when I run the simulatio

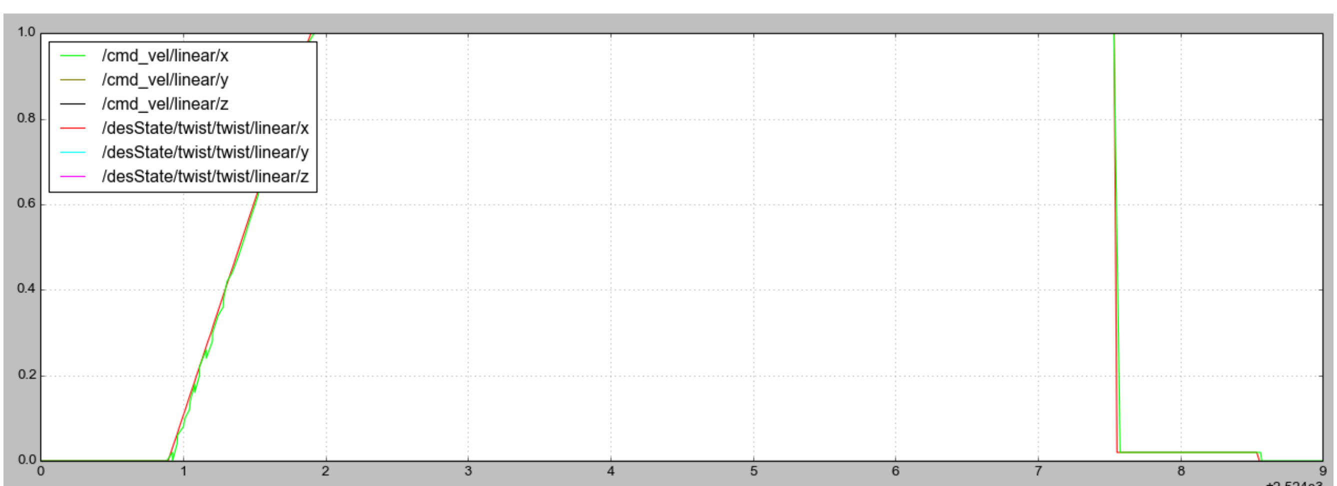


This shows that the alarm can be heard by the callBack function and can successfully call the halting function. But I can not move the robot to a place where no alarm is heard. Thus I have to use the e-stop service to test my Halting function works.

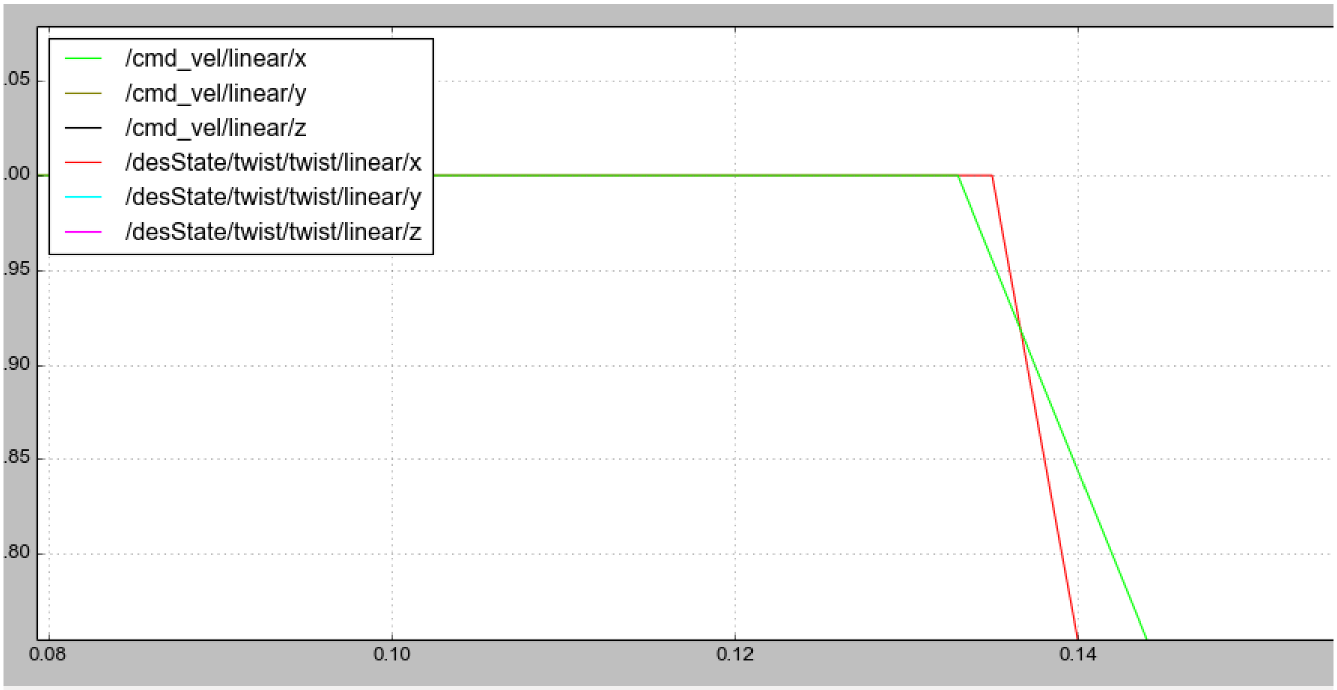
Part 3: robot Halting function test

This is test by call roseservice as indicated by Dr.Newman's write up :
`rosservice call estop_service`
`rosservice call clear_estop_service`

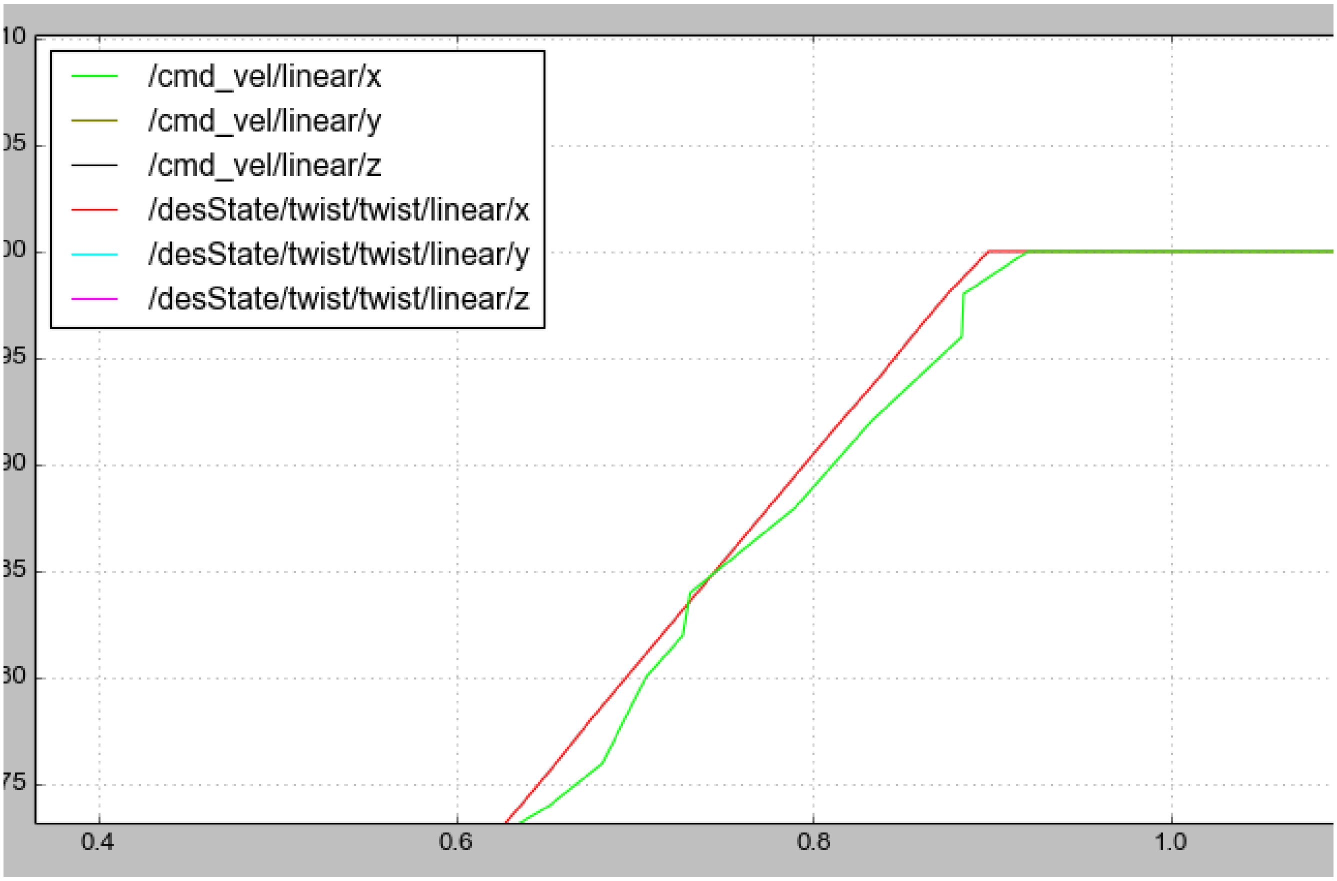
rqt plot results:



Halting:



Recover:



/cmd_vel/linear/x
/cmd_vel/linear/y
/cmd_vel/linear/z
/desState/twist/twist/linear/x
/desState/twist/twist/linear/y
/desState/twist/twist/linear/z
0.4
0.6
0.8
1.0